A Simple Baseband Processor for RF Transceivers

Rejeesh Kutty
Analog Devices, Inc.

Introduction

Today, wireless systems are ubiquitous, and the number of wireless devices and services are continuing to grow. The design of a complete RF system is a multidisciplinary design challenge, with the analog RF front end being the most critical part of it. However, the availability of integrated RF transceivers such as AD9361 greatly reduces the RF challenges on such designs. These transceivers provide a digital interface for the analog RF signal chain and allow easy integration to an ASIC or FPGA for the baseband processing. The baseband processor (BBP) allows user data to be processed in the digital domain between an end application and the transceiver device. The baseband processor design is also easily designed using system modeling tools such as Simulink. However, a novice user may find it difficult to understand and fill in this piece of the communication system puzzle. This article is a modest attempt to design and implement a simple RF baseband processor for an over the air communication system. The design is implemented on the AD-FMCOMMS2-EBZ and Xilinx® ZC706 platform using the AD9361 FPGA reference design framework.

The first section of this article details the general design principles of this baseband processor. This section is mostly a theoretical introduction of the BBP. In the second section, the actual hardware implementation of the BBP is discussed using the AD9361 FPGA reference design from Analog Devices. It is noted that the main design goal is to keep the design as simple as possible and demonstrate a quick over the air data transfer in a lab environment. There are regulations and other implications in using and, thus, interfering with the RF spectrum.

Base Design

A typical RF system is shown in Figure 1, with the exception of direct RF systems. Only a single data path is shown in this Figure 1, the reverse direction is the mirror image of this data path. The baseband processor in question and presented in this article allows data to be processed in such a way that it is transferred over the air between the RF two systems. The base design requirements are discussed below.

Data Is Repeated on Both the Orthogonal Signals I&Q

Note that the carriers are independent and asynchronous to each other. Thus, there are phase and frequency offsets between transmit and receive carriers. This has an adverse effect on the demodulation at the receiver. A significant problem is signal inversion, the orthogonal signals may reverse their roles as the offsets periodically merge and drift apart. A simple method to overcome this ambiguity is to repeat the same data on both the orthogonal signals.

Data Is Transmitted and Received Serially (Bit-Wise)

The RF front-end interface to the BBP in most cases is a DAC and an ADC. These are the digital interfaces of an analog signal. Thus, it is not possible to simply send the data to the DAC input and expect the same at the ADC output. The data is transmitted serially, mapping the single bit data to the full resolution of the DAC. Similarly, the data is received serially, demapped from the full resolution of the ADC. This provides an ample amount of redundancy. If these were 16-bit converters, the receiver would make a decision of a 1 or 0 from a possible 65536 data set. This alone simplifies the decoding significantly.

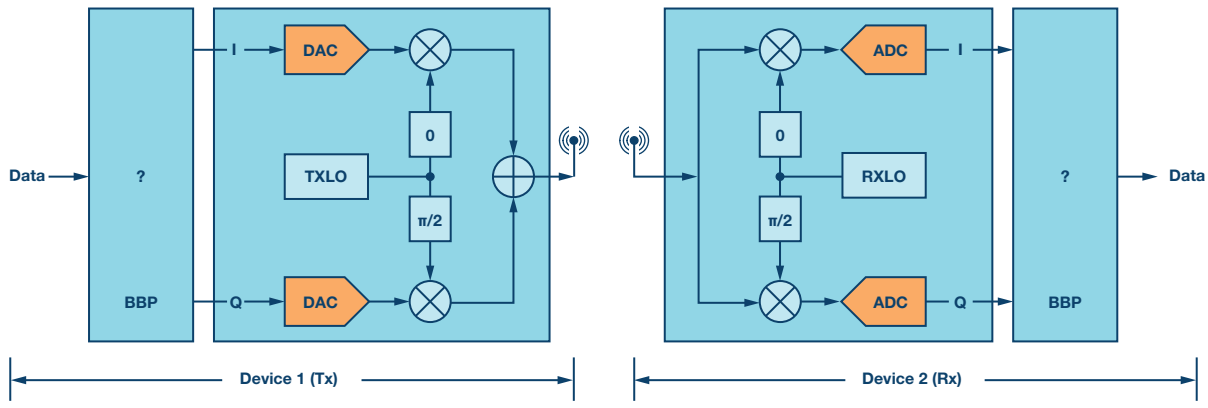

Figure 1. RF system block diagram.

I&Q Signals Are Orthogonal to Each Other

The RF front-end device (such as AD9361) is an I/Q transceiver. These devices work the best if the inputs are orthogonal signals. These devices usually feature internal I/Q matching and correction along the two data paths so as to offset any variations between them. The convention is that the real (I) signal is a cosine function and the imaginary (Q) signal is a sine function.

The Modulation Scheme Is BPSK

It is possible to deploy all of the commonly known methods, amplitude, frequency, or phase modulation of the signals. It is relatively easier to detect phase differences. Since data is transferred serially, the natural choice is binary phase shift keying (BPSK).

The Bit Interval Is Eight Samples

The data need timing information, the bit interval. The maximum possible bit interval is the sampling period. In order to keep the receiver simple, it requires ample time to decode the signal and make a decision. The simplest methods of timing recovery are zero crossing and peak detection. In this case the peaks are not going to be consistent. So zero crossing is chosen for detecting and tracking bit intervals. There are also carrier differences between the two systems. In some cases, a sample may be ambiguous at either end of the user data. Allowing four samples for each half of the sinusoidal signal, the bit interval is set to eight samples. Thus, the effective transfer rate is the sampling frequency divided by eight.

Data Has No DC Content

The timing and relative phase recovery is based on zero crossing of the signals. Thus, the individual signals need to be free from any dc content. It also requires the signal to allow at least one zero crossing every bit interval. A sinusoidal signal has both these properties and nicely fits with the BPSK modulation scheme mentioned above.

Data Is Scrambled

The user data is arbitrary—it may very well be a long sequence of 1s or 0s. The data need to be scrambled to allow the timing and phase recovery at the receiver to track the signal more effectively.

Data Is Transferred in Packets

The signals at the receiver are expected to have amplitude, frequency, and phase errors given that the systems are asynchronous to each other. The demodulated signal is a variation of the phase of the transmitted signal with respect to the local carrier. The carriers may track for a while, take a data hit, and track again. Thus, the design needs to be prepared to take some data loss. In order to support this, data is transferred in packets. A few packets may be repeated in lieu of the entire data.

Packets Are Validated Using CRC

The packets carry a cyclic redundancy check (CRC) so that the receiver is allowed to drop the packet if there is a mismatch and request it to be sent again.

Timing and Phase Correction Is Done During Every Preamble

The packet header carries a preamble for delineation of it from the received data stream. This preamble is also used by the receiver to reset the timing and phase information of the signals to demodulate the packet data.

Built-In Performance Metrics

The receiver also supports statistical counters such as the number of packets received, dropped, or corrected. These counters are used to measure and monitor the performance metrics including bit error rate and effective data rate.

In summary, the data is transmitted and received serially as packets. The packets carry a preamble and CRC. The data is BPSK modulated and demodulated on intermediate orthogonal signals before the transceiver device. The intermediate signal frequency and, hence, the bit rate of the data is one eighth the sampling rate. The baseband processor module with the design details outlined above is shown in Figures 2 and 3.

The transmitter reads bytes of data (character width) and converts them to packets with a header or preamble. A CRC is added to the tail end of the packet. The packet data is then scrambled and serialized. The single bit data then phase modulates a cosine (I) and sine (Q) function before interfacing to the transceiver.

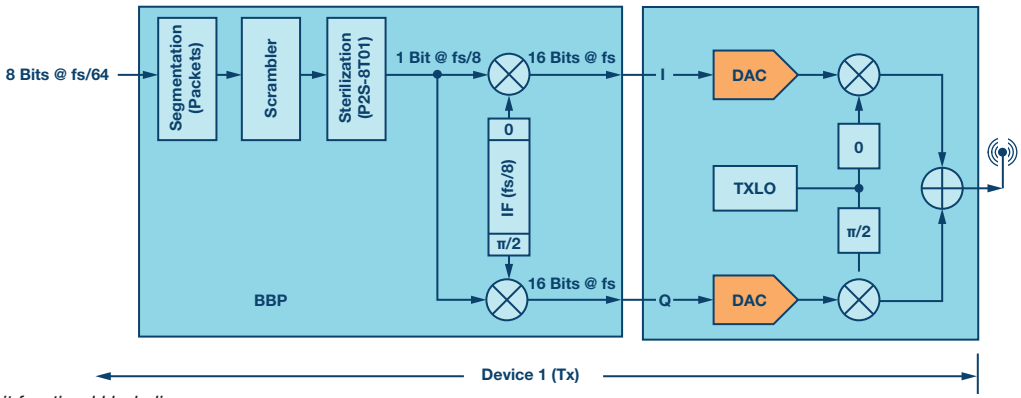

Figure 2. BBP transmit functional block diagram.

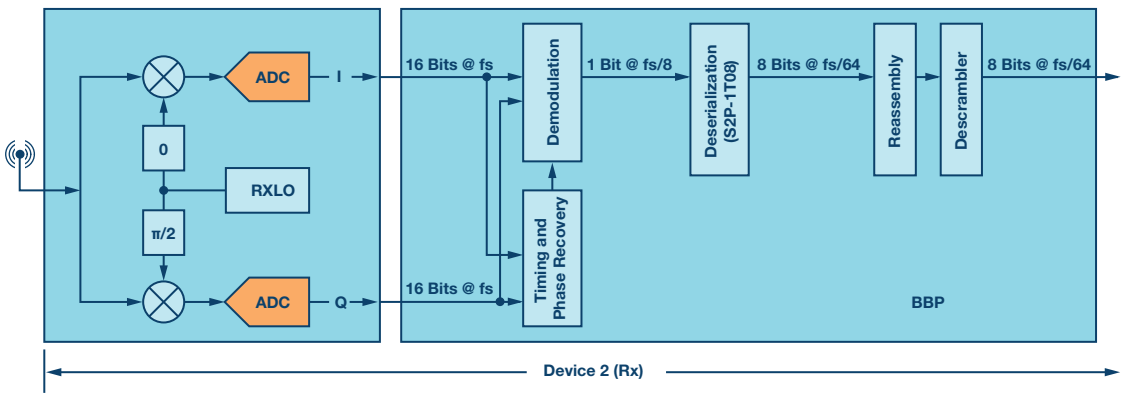

Figure 3. BBP receive functional block diagram.

In the receive direction, an offline module recovers and tracks the timing intervals and the relative phase of the modulated signal. This information is used to recover serial data from the incoming ADC samples. They are then assembled to packets and descrambled. At the end of the packet, the CRC is compared and if there is a mismatch the packet is dropped. If the CRC matches, data is passed to the end user.

Implementation

The BBP design is implemented and tested in hardware. The hardware is a combination of two evaluation boards: the Xilinx ZC706 evaluation board featuring the Zynq FPGA device and the AD-FMCOMMS3-EBZ evaluation board featuring the AD9361 transceiver. ADI provides a complete reference design supporting this hardware. This open-source design is freely available, fully supported, and updated across major tool versions. The hardware details are found in the following URLs:

ZC706

AD-FMCOMMS3-EBZ

Zynq SOC

AD9361

ADI GitHub Repository

ZC706 and AD-FMCOMMS3-EBZ HDL Reference Design

AXI AD9361 IP

The ADI reference design is an embedded system supporting the Linux® framework. It consists of various peripherals around the ARM® processors. The AD9361 device interfaces to the `axi_AD9361` IP peripheral. It transfers raw sampling data between the RF device and system memory. The peripherals and devices are initialized and controlled via Linux kernel drivers. The BBP is implemented as another IP peripheral that interfaces to the `axi_AD9361`. The BBP IP is named `axi_xcomm2ip` for historical reasons. A user space application in Linux is used to control, send, and receive data between the systems.

In the ADI reference design, the `axi_AD9361` IP interfaces to an unpack module (`util_upack`) in the transmit direction and a pack module (`util_cpack`) in the receive direction. In the transmit direction, the BBP data is inserted between the unpack module and the AD9361 core. In order for this to not affect the default data path, the BBP supports an optional data path multiplexer to select either the unpacked data source or the BBP data source. The BBP allows the reference design data path as the default and selects the BBP data source only when enabled. In the receive direction, the BBP simply interfaces to the AD9361 core. The reference design data path is unaffected. This allows the frame work to boot and set up the system unhindered. After the system setup, BBP is enabled to allow data transfer by overriding the default data path. A block diagram of the BBP as it is implemented in the ADI reference design is shown in Figure 4.

The design, initialization, and data transfer discussed in this article uses a pair of this hardware. The setup only requires a pair of HDMI® monitors, a keyboard and mouse, and antennae. The systems are completely asynchronous to each other but do require the same settings. The data is transferred on different carriers in each direction. The transmit carrier frequency of device 1 and the receive carrier frequency of device 2 are the same but different in the other direction. However, if using a single device in loopback, the transmit and receive carriers must be of the same frequency. The HDL design of the BBP utilizes ADI library modules.

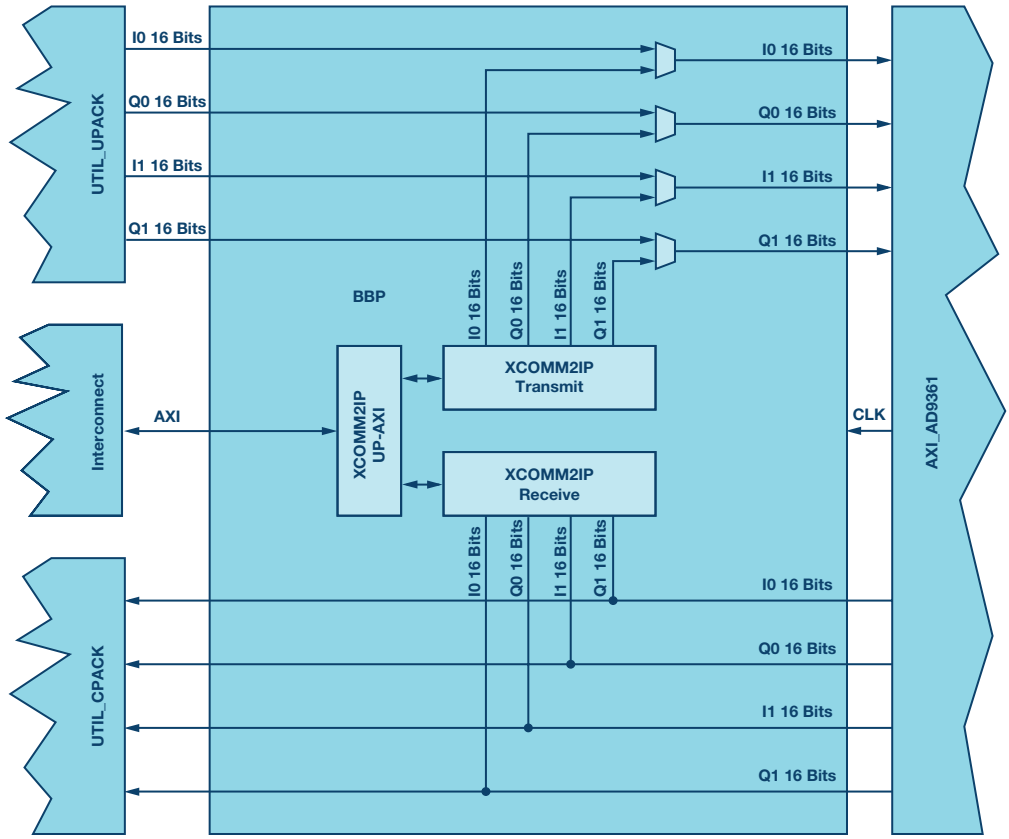

Figure 4. BBP IP block diagram.

Table 1. BBP Register Map

Address	Register Name			Type
	Fields	*Name*	*Description*	*Default*
0x000	XCOMM2IP_VERSION			RO
	31:0	VERSION	The IP version.	0x40063
0x008	XCOMM2IP_SCRATCH			RW
	31:0	SCRATCH	Scratch register.	0x0
0x800	XCOMM2IP_TX_RESET			RW
	0	TX_RESET	If set to 0x1, transmit is reset. This bit must be set to 0x0 for normal operation.	0x1
0x804	XCOMM2IP_TX_ENABLE			RW
	0	TX_ENABLE	If set to 0x0, data path is unaffected and UNPACK data is transmitted to the DAC. If set to 0x1, the BBP data is transmitted to the DAC.	0x0
0x808	XCOMM2IP_TX_REQ			RW
	0	TX_REQ	If set by software, initiates a packet transfer in the transmit direction. This bit is self-cleared by hardware when the transfer is complete.	0x0
0x80C-0x87C	XCOMM2IP_TX_PKT_DATA_3-XCOMM2IP_TX_PKT_DATA_31			W
	7:0	TX_PKT_DATA_3- TX_PKT_DATA_31	The packet data bytes 3 to 31. The hardware uses the first three bytes for header and the last byte for CRC.	W
0xC00	XCOMM2IP_RX_RESET			RW
	0	RX_RESET	If set to 0x1, receive is reset. This bit must be set to 0x0 for normal operation.	0x1
0xC08	XCOMM2IP_RX_REQ			RW
	0	RX_REQ	If set by hardware, this indicates a packet is received and needs to be read by software. The software must clear this bit after reading the packet data. All subsequent packets are dropped until this bit is cleared by software.	0x0
0xC0C-0xC7C	XCOMM2IP_RX_PKT_DATA_3-XCOMM2IP_RX_PKT_DATA_31			W
	7:0	RX_PKT_DATA_3- RX_PKT_DATA_31	The packet data bytes 3 to 31. The hardware uses the first three bytes for header and the last byte for CRC.	W

Control (Microprocessor) Interface

An AXI-Lite interface is used for controlling and monitoring the BBP by the processor. This interface module is simply inferred using the **up_axi** module from the ADI common library **(hdl/library/common/up_axi.v)**. This module translates the AXI-Lite interface to a simple memory like read and write bus. The internal registers and memory are added just like any other ADI IP. The register map is detailed in Table 1.

The **up_axi** module ports and their port mappings are described below.

up_rstn: AXI interface reset (asynchronous active low), connected to **s_axi_aresetn**.
up_clk: AXI interface clock, connected to **s_axi_aclk**.
up_axi_*: AXI interface signals, connected to equivalent **s_axi_*** ports.

up_wreq, up_waddr, up_wdata, up_wack: The internal write interface, the **up_wreq** signal is asserted to indicate a write request along with the address and data. The request needs to be acknowledged via the **up_wack** port.

A simple register write is implemented as follows.

```
always @(negedge up_rstn or posedge up_clk)
begin
if (up_rstn == 0) begin
up_wack <= 'd0;
up_reg0 <= UP_REG0_RESET_VALUE;
end else begin
up_wack <= up_wreq_s;
if ((up_wreq_s == 1'b1) && (up_waddr == UP_
REG0_ADDRESS)) begin
up_reg0 <= up_wdata[UP_REG0_WIDTH-1:0];
end
end
end
```

The module performs an address translation in between. The AXI interface uses byte addresses, but the internal bus uses DWORD addresses. The result is that the **up_axi** module drops the two least significant bits of the AXI address to generate the internal DWORD address.

up_rreq, up_raddr, up_rdata, up_rack: The internal read interface, the **up_rreq** signal, is asserted to indicate a read request along with the address. The request needs to be acknowledged via the **up_rack** port along with the read data.

The same register above implemented for read is shown below.

```
always @(negedge up_rstn or posedge up_clk)
begin
if (up_rstn == 0) begin
up_rack <= 'd0;
up_rdata <= 'd0;
end else begin
up_rack <= up_rreq_s;
if ((up_rreq_s == 1'b1) && (up_raddr == UP_
REG0_ADDRESS)) begin
up_rdata <= up_reg0;
end else begin
up_rdata <= 32'd0;
end
end
end
```

The same address translation applies for read also. The read data is only driven when requested and otherwise set to zero. This is because the **up_axi** module passes individual read data from various address blocks to an OR gate. Thus, the blocks that are not selected need to drive the read data zero.

The BBP has three address spaces as listed in the register map table above. The common register space is mapped to 0x000, transmit (DAC) is mapped to 0x800 (0x200), and receive (ADC) is mapped to 0xC00 (0x300). The software (Linux user space application) is expected to write the transmit packet data to a buffer and read the received packet data from another buffer. The packet size is chosen to be 32 bytes with a 3 byte preamble and a 1 byte CRC.

Data Interface

The AD9361 interface core consists of two pairs of 16-bit I/Q data for the two channels in receive and transmit direction. The core runs at the same clock as the AD9361 digital interface. In 2R2T mode, this is 4× the sampling rate. In 1R1T mode, this is 2× the sampling rate. The effective data rate is controlled by the valid signal. Thus in 2R2T mode, valid is asserted once every four clocks. In 1R1T mode, valid is asserted every two clocks. The BBP is designed to support both 2R2T and 1R1T mode. It uses a single transmit and receive channel. The internal logic is made to run at the sampling rate in both 2R2T and 1R1T mode. The BBP then transfers data with the interface core at its clock frequency. This is intentionally done to demonstrate clock conversion within the BBP. In many cases a user may want to run the BBP logic at the sampling rate regardless of the interface rate of the transceiver.

The internal clock at the sampling frequency is generated using Xilinx primitives BUFR and BUFG. The BUFR is a divider and BUFG is a high fan-out clock buffer. It is also possible to use a MMCM for this purpose. The internal clock is generated as follows.

```
parameter XCOMM2IP_1T1R_OR_2T2R_N = 0;
localparam XCOMM2IP_SCLK_DIVIDE =
(XCOMM2IP_1T1R_OR_2T2R_N == 1) ? "2" : "4";

BUFR #(.BUFR_DIVIDE(XCOMM2IP_SCLK_DIVIDE))
i_bufr (
 .CLR (1'b0),
 .CE (1'b1),
 .I (clk),
 .O (s_clk_s));

BUFG i_bufg (
 .I (s_clk_s),
 .O (s_clk));
```

The use of BUFR and BUFG ensures that the clocks are frequency locked at the expense of phase certainty. The maximum phase ambiguity is a single period of the interface clock. This is easily compensated by a four stage register array with a synchronization signal. However, the design uses dual port RAM modules to implement the data transfer. This is intentionally done as a use case example of common signal processing requirements. The dual port RAM elements are inferred using the ADI library memory modules (**ad_mem**).

Transmit Interface

In the transmit direction, the processor writes packet data to a buffer (see the register map table above). It then requests to the hardware to send this packet. The BBP continuously sends packets to the device. At the beginning of a packet, it checks for any requests. If there is no pending request, it transmits an idle packet. If a request is pending, the packet buffer is read and transmitted.

The transmit logic runs at bit width using a free running bit counter. The buffer read address is updated when the bit counter is 0x0. Since the processor request may occur any time during a packet transfer, it is captured immediately and cleared at the beginning of a packet transfer. At the beginning of a packet transfer, if a request is pending it is acknowledged back to the processor interface. The request is used to select between buffer data or idle data.

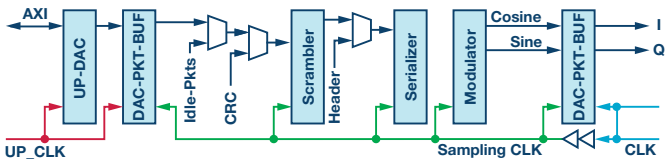

Figure 5. Transmit data path.

The first two bytes of the packet data is set to 0xfff0. The third byte is used to indicate an idle (0xc5) or a data (0xa6) packet. The CRC byte is inserted as the last byte of the packet. The CRC polynomial is x8 + x2 + x + 1. All the bytes except the header are scrambled. The scrambling polynomial is same as SONET/SDH (x7 + x6 + 1).

A cosine and sine lookup table is used to generate the modulated carrier. The bit interval equals the full cycle (0 to 2π) of the signal in eight samples. The bit data is used to invert the signal. The data is then written to a small buffer and read based on the valid signal from the AD9361 interface core using the interface clock.

Receive Interface

In the receive direction, the I/Q data is monitored for the header pattern 0xfff0. This is the unique pattern that appears once in a packet transfer. It is possible to send packet data such that the scrambler output repeats this pattern. This is discouraged and prevented by software. This series of in-phase data sequence for 12 consecutive bit intervals is used to reset and track the receiver timing and phase by the timing recovery module. Thus, it resets its timing counters and sets its phase value to 0x1. The first inversion after this sequence is considered 0x0. After this, the timing recovery module maintains its state throughout the packet transfer.

The data recovery module averages the signal and makes a decision on the current phase of the signal. It is then compared to the relative phase tracked by the timing recovery module. In the case of a conflict, decision is based on past changes on the signal. This is because conflicts usually arise from phase switching.

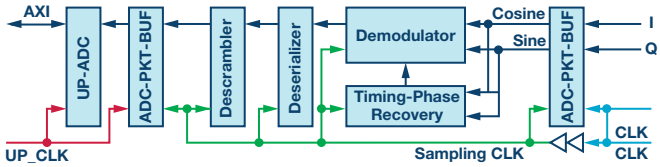

Figure 6. Receive data path.

The demodulated bit data is then assembled to bytes and descrambled. The data is written to the transfer buffer if it is empty. The CRC is verified at the end of packet. If it matches and the transfer buffer is written, the processor interface is notified. The software is expected to monitor this request and, if set, empty the buffer by reading its contents. It must then clear the request to allow further packet transfers.

Build Instructions and Downloads

This article provides the theory and implementation specifics for a simple RF baseband processor. A practical implementation of this design on the ZC706 and AD-FMCOMMS3-EBZ hardware is discussed. The complete design files for a quick demonstration and build instructions are documented at *https://wiki.analog.com/resources/fpga/docs/hdl/xcomm2ip*. The wiki also details the HDL design, software, RF setup, performance, and analysis.

About the Author

Rejeesh Kutty is an HDL engineer in the Customer Solution Enablement Group at Analog Devices and he joined the company in 2011. He received a master's degree in electronic engineering from Indian Institute of Science, India.

Online Support Community

EngineerZone™
SUPPORT COMMUNITY

Engage with the Analog Devices technology experts in our online support community. Ask your tough design questions, browse FAQs, or join a conversation.

Visit *ez.analog.com*

Analog Devices, Inc.
Worldwide Headquarters

Analog Devices, Inc.
One Technology Way
P.O. Box 9106
Norwood, MA 02062-9106
U.S.A.
Tel: 781.329.4700
(800.262.5643, U.S.A. only)
Fax: 781.461.3113

Analog Devices GmbH
Europe Headquarters

Analog Devices GmbH
Otl-Aicher-Str. 60-64
80807 München
Germany
Tel: 49.89.76903.0
Fax: 49.89.76903.157

Analog Devices, Inc.
Japan Headquarters

Analog Devices, KK
New Pier Takeshiba
South Tower Building
1-16-1 Kaigan, Minato-ku,
Tokyo, 105-6891
Japan
Tel: 813.5402.8200
Fax: 813.5402.1064

Analog Devices, Inc.
Asia Pacific Headquarters

Analog Devices
5F, Sandhill Plaza
2290 Zuchongzhi Road
Zhangjiang Hi-Tech Park
Pudong New District
Shanghai, China 201203
Tel: 86.21.2320.8000
Fax: 86.21.2320.8222

TA14701-0-8/16

analog.com

ANALOG DEVICES
AHEAD OF WHAT'S POSSIBLE™